PROGRAMACIÓN CON RESTRICCIONES EN PROBLEMAS DE SCHEDULING: PLANIFICACIÓN DE ACTIVIDADES Y RECURSOS DE PRODUCCIÓN

Francisco S. Ibáñez, Daniel Díaz Araya y Raymundo Q. Forradellas
Laboratorio Integrado de Sistemas Inteligentes
Instituto de Informática
Universidad Nacional de San Juan,
e-mail: {fibanez, ddiaz, kike}@iinfo.unsj.edu.ar

Este trabajo está desarrollado en el marco del proyecto titulado "Problemas de asignación de recursos utilizando Técnicas de Inteligencia Artificial"

1 Introducción

En este trabajo, se presentan los aspectos más importantes de la representación de un problema de scheduling complejo, modelado mediante un enfoque basado en restricciones.
La metodología usada combina las ventajas de la programación basada en restricciones con las de la programación orientada a objetos, siendo posible definir entre otras cosas, actividades, recursos y restricciones, entre actividades por un lado, y por otro lado entre actividades y recursos.
Inicialmente se describe el problema a ser resuelto. A los efectos de enfatizar los aspectos más relevantes del uso de la metodología, deliberadamente se han desconsiderado algunos aspectos del problema.
Una vez descripto el problema a resolver, se presentan sucintamente los conceptos más relevantes de la metodología utilizada para resolver el problema de aplicación.
Finalmente, se expone la utilización de la metodología, para resolver el problema de aplicación propuesto.

2 Descripción del problema

La entrada al problema está constituida esencialmente por un conjunto de *pedidos* de productos finales, efectuados por clientes de una empresa que produce el material necesario para fabricar envases de diferentes tipos de productos. A partir de estos pedidos se debe determinar un conjunto de *actividades* que necesitan llevarse a cabo a los efectos de obtener los productos finales requeridos por los clientes.
El objetivo de la aplicación es desarrollar una planificación de actividades y de recursos, de modo tal que se verifiquen todas las restricciones.

2.1 Actividades

Los procesos básicos necesarios para la obtención de un producto final involucra las siguientes actividades principales:

Este trabajo fue desarrollado en el marco del Proyecto CICITCA (UNSJ) "Problemas de Asignación de Recursos utilizando Técnicas de Inteligencia Artificial"

- Impresión
- Laminado
- Corte
- Empaque

Estas actividades deben realizarse en un orden específico. Según el producto final que se desea obtener, estas actividades pueden usar diferentes máquinas. La actividad de Impresión puede utilizar una máquina Hueco-Impresora o Flexo-Impresora.
La actividad de Laminado puede utilizar tres tipos diferentes de máquinas: Máquina de Laminado por Cera, por Solvente, o por Termofusión.
La actividad de Corte puede utilizar dos tipos de máquinas diferentes. Ambos tipos de máquinas realizan el mismo trabajo, la diferencia sólo radica en la velocidad de proceso.
Finalmente, la actividad de Empaque requiere sólo recursos humanos.
Las actividades, además de requerir máquinas (o recursos humanos) para poder llevarse a cabo, requieren recursos. Estos recursos, en general, pueden ser insumos (bobina con la materia prima), o productos semielaborados (por ejemplo, bobinas impresas).
Una vez finalizadas las actividades, estas producen recursos que son utilizados por las actividades siguientes, o constituyen el producto final. Por ejemplo, la actividad de Laminado, requiere bobinas impresas (obtenidas por la actividad de Impresión), y una vez finalizada, produce bobinas laminadas (que serán utilizadas por la actividad siguiente).
La actividad de impresión, requiere dos tipos de recursos, además de la máquina que realiza la actividad. Por un lado, requiere insumos (bobina inicial), y por otro lado, requiere rodillos grabados, para efectuar la impresión.
Para obtener un producto terminado, es necesario ejecutar una secuencia de actividades, en un determinado orden, teniendo en cuenta las máquinas particulares que se utilicen, y considerando además los recursos que se requieran, antes de comenzar cada actividad, y los recursos producidos, luego de la finalización de las mismas.

2.2 Trabajos (Jobs)

Un Trabajo, está constituido por la descripción de todas las actividades necesarias para obtener un producto final, y de las restricciones de precedencia y de requerimientos de recursos de las mismas.
Para un trabajo particular, puede que ya exista un rodillo grabado, necesario para realizar la actividad de impresión. En este caso no será necesario realizar la actividad de Grabación para producirlo. La cantidad de producto final que se obtiene para un trabajo determinado, se mide en Kilogramos. Un trabajo particular, sólo puede obtener una cantidad limitada de producto terminado. A partir de un pedido, efectuado por un cliente, se debe determinar, en primer lugar, los trabajos necesarios para satisfacer el pedido. Por ejemplo, si una firma particular desea obtener 2500 Kg de producto final, y cada trabajo (debido a las limitaciones de las máquinas que utiliza) puede producir 1000 Kg como máximo, serán necesarios tres trabajos para completar el pedido.
Para cada pedido, existe una cantidad de rodillos ya grabados, y por lo tanto, los trabajos pueden o no incluir la grabación de rodillos.
La salida del problema, está constituida por una asignación de tiempos de comienzo y de finalización de cada una de las actividades involucradas en la representación del problema. El problema ha sido resuelto optimizando el tiempo de finalización del proyecto, para algunos casos, y conformándose con tiempos subóptimos para casos más complejos, sin embargo, en el presente trabajo, sólo consideraremos los aspectos más relevantes de la representación del problema.

3 Descripción de la Metodología usada.

En programación con restricciones, un problema se representa esencialmente, en términos de variables que involucran incertidumbre, y restricciones que deben verificar estas variables. En esta aproximación se usa diseño orientada a objetos [Booch, 91] para representar las variables del problema.

Una vez representado el problema, la resolución del mismo consiste en encontrar valores de cada una de las variables, de modo que se verifiquen todas las restricciones [Ddiaz,98]. Esta actividad se conoce como búsqueda de soluciones, puesto que de antemano no se sabe si existe o no una solución.

Finalmente, se debe considerar el criterio de optimización para elegir una solución. Un criterio de optimización usado muy frecuentemente es minimizar el tiempo de finalización del proyecto.

En el problema particular que estamos considerando, será necesario representar pedidos, trabajos (jobs), actividades, recursos y restricciones, por lo tanto, en el presente trabajo, consideraremos solamente aquellos conceptos de la herramienta propuesta, relacionados con el problema a tratar.

La parte más compleja de la modelización, consiste en establecer apropiadamente la relación entre actividades, recursos y restricciones.

Las actividades son realizadas mediante la utilización de máquinas. Una de las principales ventajas de los mecanismos basados en restricciones, es que mediante la utilización de los mismos, es posible representar explícitamente un problema de scheduling. Tal representación sirve además como una especificación declarativa. Finalmente, a la hora de resolver un problema, es posible confiar en las primitivas de control del resolvedor de restricciones, sin tener necesidad de conocer cómo trabaja internamente el mecanismo de propagación de restricciones.

En este apartado usaremos una sintaxis basada en [I,Sch-R,95], [I,Sch-U,95], aunque los conceptos sbyacentes a este trabajo no se restringen a ser implementados con esta sintaxis.

Un problema de scheduling, se puede representar como una instancia de una clase: *IlcSchedule* y consiste en un conjunto de actividades, cuyas ejecuciones requieren o proveen recursos.

Una tal instancia, involucra un origen (mínimo tiempo en que pueden comenzar las actividades) y un horizonte (máximo tiempo en que pueden terminar las mismas). Las actividades que forman parte de un schedule, se ejecutarán dentro del intervalo de tiempo determinado por el origen y el horizonte, y las restricciones sobre recursos, se aplicarán exclusivamente dentro de este intervalo.

La declaración

IlcSchedule sched(tiempoMin, tiempoMax)

crea la instancia *sched* de la clase *IlcSchedule,* con *tiempoMin* y *tiempoMax* como el origen y el horizonte, respectivamente.

3.1 Actividades

Una actividad se representa como una instancia de la clase *IlcIntervalActivity.* Una instancia de esta clase representa una actividad que se ejecuta sin interrupción desde su tiempo de comienzo hasta su fin.

Para crear una tal instancia, es necesario especificar el schedule (formalmente, la instancia de la clase *IlcSchedule*) al que pertenece, y la duración de la actividad que representa.

La declaración

IlcIntervalActivity act(sched, dur)

genera la instancia *act* de la clase *IlcIntervalActivity,* que pertenece al schedule *sched*, y que representa una actividad cuya duración es *dur*.

3.2 Recursos

Las actividades normalmente usan recursos para poder llevarse a cabo, y además pueden producir recursos.

Puede ocurrir que sólo haya un único recurso de un tipo determinado, en cuyo caso se dice que el recurso es *unario*. Si existe más de un recurso con las mismas características, de modo tal que sea indistinto cuál se utilice, o bien, existe un solo recurso medible en unidades discretas, se utiliza un recurso *discreto* para modelar tal situación.

Un recurso unario se representa como una instancia de la clase *IlcUnaryResource.* En la misma debe especificarse el schedule al que está ligado, y debe determinarse si el recurso es requerido o provisto.

Por ejemplo, la declaración:

IlcUnaryResource rec(sched, IlcRequeridedResource)

genera la instancia *rec* de la clase *IlcUnaryResource,* que pertenece al schedule *sched*, y que representa un recurso requerido.

Para el caso de recursos discretos, se debe especificar además, la capacidad máxima del recurso.

Por ejemplo, la declaración

IlcDiscreteResource rec(sched, IlcRequeridedResource, maxCap)

genera la instancia *rec* de la clase *IlcDiscreteResource,* que pertenece al schedule *sched*, y que representa un recurso requerido de capacidad máxima *maxCap*

3.3 Restricciones

Existen fundamentalmente dos tipos de restricciones que relacionan actividades entre si:

- Restricciones temporales
- Restricciones sobre recursos

3.3.1 Restricciones temporales

Mediante una restricción temporal, es posible especificar las siguientes relaciones entre actividades:

- Una actividad debe comenzar después de que comience otra actividad.
- Una actividad debe terminar después de que comience otra actividad.
- Una actividad debe comenzar después de que finalice otra actividad.
- Una actividad debe terminar después de que finalice otra actividad.

Las restricciones temporales sobre actividades, se representan mediante funciones miembros de la clase *IlcIntervalActivity.*

Por ejemplo, para representar restricciones de precedencia, se usa la función miembro *StartsAfterEnd* de la clase *IlcIntervalActivity.* Las restricciones se incluyen en el resolvedor de restricciones mediante la primitiva *IlcTell.*

Por ejemplo, creadas dos instancias *act1* y *act2*, de la clase *IlcIntervalActivity*, que representan dos actividades, la instrucción:

IlcTell(act1. StartsAfterEnd(act2))

establece que *act1* debe realizarse después de que haya finalizado la actividad *act2*.

3.3.2 Restricciones sobre recursos

Las restricciones sobre recursos son más complejas de comprender en profundidad.
Una actividad puede requerir o proveer un recurso. La forma general para especificar que una actividad requiere un recurso discreto (en cantidad menor o igual que la capacidad máxima) es la siguiente:

IlcTell(act.requires(rec, n))

Esta restricción establece que la actividad *act* requiere *n* unidades del recurso *rec*.
De forma similar, la forma general para especificar que una actividad provee un recurso discreto es la siguiente:

IlcTell(act.provides(rec, n))

Esta restricción establece que la actividad *act* provee el recurso *rec* en cantidad *n*, durante su ejecución.

4 Aplicación de la metodología para modelar el problema propuesto

Como se comentó en el apartado 3, la resolución del un problema de scheduling se divide básicamente en tres partes, la representación del problema, la búsqueda de soluciones, y el criterio de optimización. En el presente trabajo, consideraremos sólo la primera parte.
En la representación debemos incluir todas las variables involucradas en el problema, tanto aquellas que representan actividades como recursos.
En esta aproximación usamos programación orientada a objetos. Sin embargo, antes de definir las clases y las instancias de las mismas, es necesario reflexionar en algunas consideraciones relevantes en cuanto al diseño del modelo.
En primer lugar es importante distinguir la necesidad o no de usar recursos.
Las actividades que usan máquinas para llevarse a cabo, necesariamente deben modelar las máquinas como recursos y deben además requerir los mismos.
Dependiendo de cuántas máquinas con las mismas características existan, los recursos para modelarlas serán unarias o discretas. Por ejemplo, la actividad de Laminado, como vimos anteriormente, puede requerir tres tipos diferentes de máquinas:

- Máquina de laminado por Cera
- Máquina de laminado por Termofusión
- Máquina de laminado por Solvente

Sin embargo, sólo existe una máquina que realiza el laminado por Cera y sólo una máquina que realiza el laminado por Termofusión, y por lo tanto, estas máquinas se modelan como recursos unarios. Por otro lado, existen dos máquinas que realizan el laminado por solvente, y por lo tanto, estas máquinas deben modelarse como un recurso discreto, de capacidad máxima igual a dos.
En todos los casos, las actividades de laminado "usan" una determinada máquina, sea de laminado por cera, por solvente, o por termofusión, y consecuentemente, los recursos correspondientes han de ser requeridos y las restricciones deberán ser de requerimiento de los mismos.

4.1 Modelado de las actividades de grabación y de impresión.

Tanto las actividades de grabación como las de impresión, requieren máquinas para llevarse a cabo, y de un modo similar al caso del laminado, estas máquinas deben modelarse como recursos requeridos.
Para el caso de la actividad de impresión, además de la máquina necesaria para llevar a cabo la actividad, es necesario disponer de rodillos grabados, a los efectos de efectuar la impresión en una bobina virgen.

Como se mencionó anteriormente, estos rodillos pueden o no estar ya grabados. En el caso de que no hayan rodillos grabados disponibles, antes de llevar a cabo la actividad de impresión, es necesario previamente realizar la actividad de grabación de rodillos.
Para modelar esta dependencia de actividades, es necesario profundizar más en el concepto de recursos requeridos y recursos provistos.

4.1.1 Modelado de rodillos grabados como recursos requeridos y provistos

Las actividades de grabación proveen rodillos grabados, y por lo tanto, estos deben modelarse como recursos provistos. La actividad de impresión requiere rodillos grabados, y consecuentemente, los rodillos que se vayan a utilizar para la actividad de impresión deben modelarse como recursos requeridos.
Físicamente , los rodillos provistos por las actividades de grabación y los requeridos por las actividades de impresión son los mismos, sin embargo, debido a que un recurso no puede declararse como provisto y requerido a la vez, es necesario usar dos tipos de recursos diferentes para representar los mismos rodillos, una instancia de un recurso provisto y otra de un recurso requerido, y debido a que, conceptualmente, estas instancias representan el mismo recurso, se debe incluir una restricción que establece que en todo instante de tiempo, la cantidad de requerimientos (en cantidad 1) del recurso requerido no puede superar la cantidad provista del recurso declarado como provisto.
Para contemplar el caso de que ya hayan rodillos grabados, el modelo agrega una actividad ficticia de duración igual a cero, que provee tanta cantidad de rodillos grabados como rodillos disponibles haya.

4.2 Modelado de insumos, productos semielaborados y productos finales

Las actividades en general, además de necesitar máquinas para poder llevarse a cabo, requieren insumos o productos semielaborados.
Por ejemplo, la actividad de impresión, requiere una bobina de un determinado material como insumo, y al finalizar la actividad produce una bobina impresa, que usualmente se denomina producto semielaborado. La actividad de laminado, comienza utilizando la bobina impresa, y produce una bobina impresa y laminada.
Normalmente, los insumos, para el caso de la actividad de impresión, y por otro lado, los productos semielaborados necesarios para comenzar las actividades restantes, deberían modelarse como recursos requeridos, mientras que, por otro lado, el producto final producido por la actividad de empaque, y los productos semielaborados producidos por las actividades anteriores, deberían modelarse como recursos provistos. Sin embargo, para la aplicación que estamos considerando, no es necesario la utilización de recursos para modelar la dependencia de actividades.
Como vimos anteriormente, a partir de los pedidos de los clientes, se determinan los trabajos (Jobs) necesarios para satisfacer los pedidos. Para cada trabajo, se conoce a priori, la secuencia de actividades necesarias para completar el mismo. Suponiendo que un pedido de un cliente, determine la necesidad de realizar n trabajos, y asumiendo que Impresión_i, Laminado_i, Corte_i, Empaque_i, con $1 \leq i \leq n$ es la secuencia de actividades, necesaria para completar el i-ésimo trabajo, y contando con bobinas vírgenes suficientes para las actividades de impresión, sólo son necesarias restricciones temporales para modelar la dependencia de actividades.
En efecto, dado que los productos semielaborados pasan de una máquina a la siguiente, dentro de un mismo trabajo, el producto semielaborado producido por una actividad, y necesario antes de comenzar la actividad siguiente, sólo impone una restricción temporal de precedencia, entre ambas actividades.

Si el producto semielaborado, necesario para comenzar una actividad de un determinado trabajo, pudiera ser provisto por una actividad de cualquier otro trabajo, sería necesario la utilización de recursos requeridos y provistos para modelar el problema. Puesto que el producto semielaborado, necesario para comenzar una actividad de un trabajo, sea el producido, precisamente, por la actividad anterior correspondiente al mismo trabajo, y dado que estas actividades se conocen a priori, basta incluir sólo restricciones temporales de precedencia para establecer correctamente las relaciones entre las actividades.

4.3 Consideración de tiempos adicionales entre actividades consecutivas

En el problema que estamos tratando, es necesario considerar tres tiempos que se detallan a continuación, inherentes a la relación temporal existente entre dos actividades consecutivas act_j y act_{j+1}:

t_1: Tiempo ocupado en retirar la bobina (producto semielaborado) de la máquina que realiza la actividad act_j.

t_2: Tiempo para trasladar la bobina a la máquina que realiza la actividad act_{j+1}.

t_3: Tiempo ocupado en colocar la bobina en la máquina que realiza la actividad act_{j+1}.

El tiempo t_2 es muy pequeño en relación a la duración de las actividades, y por lo tanto puede despreciarse.

Los tiempos t_1 y t_3, son significativos en relación a la duración de las actividades, y consecuentemente deben tenerse en cuenta.

Para el problema, motivo del presente trabajo, es suficiente considerar los tiempos t_1 y t_3 en el cálculo de las duraciones de las actividades act_j y act_{i+1}, respectivamente, para modelar correctamente el problema. Concretamente, la duración a considerar de la actividad act_j, debe ser igual a la duración de la actividad propiamente dicha, más el tiempo t_1, mientras que la duración a considerar de la actividad act_{j+1}, debe ser igual a la duración de la actividad propiamente dicha, más el tiempo t_3.

4.4 Definición de clases necesarias para modelar el problema de aplicación

Se define una clase para representar los pedidos. Cada pedido efectuado por un cliente, generará una instancia de esta clase.

Esta clase contiene datos miembros que permiten almacenar:

- la cantidad de trabajos que resultaron necesarios para completar el pedido,
- la descripción del pedido,
- la cantidad de rodillos ya grabados, disponibles para ese pedido,
- un arreglo, donde cada elemento del mismo, es una instancia de una clase que se define para representar trabajos.

El cálculo de la cantidad de trabajos se realiza en función de la cantidad de Kg del pedido y de la cantidad máxima de Kg. que puede procesar cada trabajo.

La clase que se define para representar trabajos contiene datos miembros que permiten almacenar:

- la cantidad de actividades
- una instancia de una clase que se define para representar la actividad de grabación de rodillos.
  un arreglo, donde cada elemento del mismo, es una instancia de una clase que se define para representar las actividades principales.

La clase que se define para representar la actividad de grabación de rodillos, contiene datos miembros que permiten almacenar:

- el identificador de la actividad de grabación de rodillos para el trabajo y el pedido particular que se está representando.
  la duración de la actividad de grabación de rodillos para el trabajo y el pedido considerados.
  un valor booleano que determina la necesidad o no de realizar la actividad de grabación de rodillos para el trabajo y el pedido considerados.

La clase que se define para representar las actividades principales, contiene datos miembros que permiten almacenar:

- el identificador de la actividad que se está representando.
- la duración de la actividad
- un valor booleano que determina si el recurso que utiliza esa actividad es unario o discreto.
  un identificador del recurso, que representa la máquina que usa esta actividad.
  una variable dominio que representa el comienzo, que la actividad tendrá una vez resuelto el problema.

4.5 Generación de instancias de las clase definidas

Inicialmente. se crea una instancia de la clase *IlcSchedule* mediante la declaración:

IlcSchedule sched(0, h)

donde h es un valor entero suficientemente grande que representa el horizonte.

4.5.1 Pedidos

Para cada pedido, se crea una instancia de la clase pedido, y se realizan las siguientes tareas.

Se generan dos instancias de la clase *IlcDiscretResource* para representar recursos producidos y requeridos:

IlcDiscreteResource recProv(sched, IlcProvidedResource, maxCap)
IlcDiscreteResource recRequ (sched, IlcRequiredResource, maxCap)

donde *maxCap* represena la capacidad máxima del recurso.

Se agrega una instancia de la clase *IlcIntervalActivity* de la forma

IlcIntervalActivity nula(sched, 0)

que representa la actividad nula que produce los rodillos grabados que se encuentran en existencia y se introduce una restricción de la forma:

IlcTell(nula.produces(recProv,n)))

donde *n* representa la cantidad de rodillos grabados que se encuentran en existencia para ese pedido.

Se crean tantas instancias de la clase definida para representar trabajos como cantidad de trabajos se hayan determinado para ese pedido.

4.5.2 Trabajos

Para cada instancia de la clase definida para representar trabajos se realizan las siguientes tareas:

Si el trabajo incluye la actividad de grabación de rodillos, se introduce una declaración de la forma:

IlcIntervalActivity actGrab(sched, dur)

donde *dur* representa la duración de la actividad de grabación de rodillos, y se incluye una restricción de la forma:

Ilctell(actGrab.produces(recProd,1))

que expresa que la actividad de grabación de rodillos, representada por la instancia *actGrab*, produce el recurso representado por la instancia *recProd* en cantidad igual a uno.

Finalmente, para cada actividad que constituye el trabajo, se crea una instancia de la clase que representa las actividades principales, de la forma:

IlcIntervalActivity act(sched, dur)

y se incluyen las restricciones correspondientes.

4. 6 Restricciones

Solamente para el caso de la actividad de impresión, se incluye una restricción de la forma:

IlcTell(act.requires(recRequ,1))

donde *act* es la instancia que representa la actividad de impresión, y *recRequ* la instancia que representa los rodillos grabados que serán requeridos por actividades de impresión.

Puesto que *recProd* y *recRequ* representan, conceptualmente el mismo recurso, se incluye una restricción que establece que en todo instante de tiempo, la cantidad de requerimientos (en cantidad 1) del recurso *recRequ* no puede superar la cantidad provista del recurso *RecProv.*

Para cada par de actividades consecutivas, *act1*, *act2*, se incluye una restricción de la forma:

IlcTell(act1.StartsAfterEnd(act2))

que restringe a la actividad representada por *act1* a comenzar en un tiempo posterior a la finalización de la actividad representada por *act2.*

Para cada actividad (representada por la instancia *act*), se incluye una restricción de la forma:

IlcTell(act.requires(rec,1))

donde *rec* es la instancia del recurso que representa la máquina que usa esa actividad.

5 Conclusiones

En el presente trabajo, se ha descripto un problema de scheduling, se han presentado los aspectos más relevantes de la metodología usada para resolver problemas de scheduling, y finalmente se ha aplicado esta metodología para representar el problema planteado.

La experiencia obtenida en el desarrollo de esta aplicación, podría sintetizarse esencialmente en tres puntos.

Por un lado, la resolución de este tipo de problemas es prácticamente inviable sin disponer de un resolvedor de restricciones. El resolvedor de restricciones, no necesariamente debe ser el propuesto por [I,Sol-R,95], [I,Sol-U,95], podría ser el desarrollado en el Lenguaje CHIP (Constraint Handling In Prolog) [Hente,89], usado dentro del marco de la programación lógica, y aplicado a problemas de combinatoria discreta [Rueda,95], el desarrollado en el lenguaje OZ (mOZart) [Denys ,98], usado en un marco que, además de incorporar características nuevas, conserva las características fundamentales de los paradigmas funcionales, de la programación lógica y de la programación orientada a objetos, o bien podría usarse cualquier otro enfoque basado en restricciones [Ibañez,94]. Escapa al alcance de este trabajo, un análisis comparativo de los diferentes enfoques basados en restricciones para resolver problemas de

scheduling. El resolvedor de restricciones permite, incluir de un modo declarativo las restricciones, confiando en que el mecanismo subyacente al resolvedor de restricciones realice la propagación de restricciones, eliminado las ramas del árbol de búsqueda que no conducen a solución alguna y reduciendo consecuentemente el árbol de búsqueda.
Por otro lado, mediante la utilización de la metodología utilizada, es posible distinguir claramente, la representación del problema y la resolución del mismo.
Finalmente, las ventajas proporcionadas por la programación orientada a objetos, combinadas con aquellas derivadas de la programación con restricciones, permiten desarrollar aplicaciones muy complejas en tiempos razonables. La experiencia obtenida en nuestro grupo de trabajo, con otras alternativas muy diferentes, como por ejemplo algoritmos genéticos, para resolver este tipo de problemas, no han sido muy satisfactorios, aunque si lo son para otro tipo de aplicaciones.

6 Referencias

[**Booch, 91**] Grady Booch, Object-Orient Design with Aplications, Benjamin/Cummings, Redwood City, Calif., 1991.

[**Hente,89**]. P. Van Hentenryck, "Constraint Satisfaction in Logic Programming" MIT Press, 1989.

[**Ibañez,94**] F. Ibañez, "CLP(Temp), Integración de Restricciones Temporales Métricas y Simbólicas, en el Marco CLP", Tesis Doctoral, Universidad Politécnica de Valencia, España, 1994.

[I,Sol-R,95] "Ilog Solver - Reference Manual Version 3.0", Ilog, France, 1995.

[I,Sol-U,p5] "Ilog Solver - User Manual Versión 3.0", Ilog, France, 1995.

[I,Sch-R,95] "Ilog Schedule- Reference Manual Version 2.0", Ilog, France, 1995.

[I,Sch-U,95] "Ilog Schedule - User Manual Version 2.0", Ilog, France, 1995.

[**Rueda,95**] Rueda L. Klenzi R. Gutierrez L. Ibañez F. Forradellas R., "Tratamiento de Problemas de Combinatoria Discreta mediante el Paradigma CLP", 2das. Jornadas de Inteligencia Artificial, Universidad Nacional del Sur, Bahía Blanca, 1995.

[Ddiaz,98] Daniel Diaz Araya - Aplicación de Tecnologías Basadas en Restricciones a la Resolución de Problemas Industriales -Tesis de Grado , Universidad Nacional de San Juan, 1998.

[Denys ,98] Denys Duchier, Leif Kornstaedt, Christian Schulte, and Gert Smolka. "A Higher-order Module Discipline with Separate Compilation, Dynamic Linking, and Pickling" Technical report, Programming Systems Lab, DFKI and Universität des Saarlandes, 1998.